File Transfer
Design Document

Version Number: 1.00

E2OPEN

1 Document Management

1.1 Legal Disclaimer

E2open™, its members, officers, directors, employees, or agents shall not be liable for any injury, loss, damages, financial or otherwise, arising from, related to, or caused by the use of this document or the specifications herein, as well as associated guidelines and schemas. The use of said specifications shall constitute your express consent to the foregoing exculpation.

1.2 Copyright

1.3 Trademarks

E2open is a trademark of "E2open LLC". All other product names and company logos mentioned herein are the trademarks of their respective owners. In the best effort, all terms mentioned in this document that are known to be trademarks or registered trademarks have been appropriately recognized in the first occurrence of the term.

Trademark or Registered Trademark		Company or Organization
D-U-N-S	Data Universal Numbering System	Dun & Bradstreet
GTIN	Global Trade Identification Number	EAN-UCC
GLN	Global Location Number	EAN-UCC
Rosetta Net		Rosetta Net

1.4 Prerequisites

1.5 Related Documents

References to other related documents.

Document	Location

1.6 Document Versi n History

Document Version History		
1.0	02/07/2003	Initial version – Pravin Singhal.

1.7 Document Distribution History

Document Distribution History		
01.00		QA Team, Product Management and Business Analyst
		<Component Business/Product Manager>

2 Overview

This document describes the design of File Transfer (FT) Service. The FT service is designed to efficiently handle transfer of large files over Internet. Large files cannot be transferred over Internet in a single synchronous operation reliably and efficiently. There are a number of problems that result from attempting to transmit large files in a single, synchronous operation:

1. Transmission of such a large file may consume all the available network bandwidth between the customer and E2open. This is particularly troublesome for remote locations with lightweight network connections.
2. File transfer operations that take hours to complete are prone to failure. If anything goes wrong with the network connection during the N hours it may take to transfer a large file, the entire file transfer operation may have to be aborted and re-started.

The idea behind FT service is to transmit the file in smaller manageable chunks and reassemble the chunks as they arrive at the other end. If any end becomes unavailable during a transmission or the network connection breaks, transmission resumes from where it left.

3 Highlights

The File transfer mechanism consists of 2 main components.

- **File Transfer (FT) Gateway**: The FT Gateway is installed at a customer's site.
  - Initiates an upload or download transfer request for a file.
  - Communicates with the FT Server via E2open's webseal using HTTPS.
  - Connects to E2open using user session (E2open user or CDSSO user) or can be set up to use client side certificate.
  - ***Supports multiple transfers concurrently.*** Multiple upload and download operations can proceed concurrently.
  - ***Transfers multiple chunks of the same file concurrently.*** It can also be configured to use more than one thread per transfer (preferable in the presence of a fat pipe).
  - ***Dynamically creates logical chunks*** of the file during transfer. The file is not physically chunked. Due to this
    - No extra disk space is required for temporary physical file chunks.
    - Boosts performances as no CPU or I/O time spent in creating these chunks.
  - ***Allocates the disk space for the file to be downloaded once the download transfer session is setup.*** This prevents the client from running out of disk space during the transfer.
  - Transfer session extends across both FT Client and FT Server restarts.
  - Supports both client and server initiated downloads.
    - In client initiated download (Client push), the client sends a download request and then keeps polling till the server has the file ready to be downloaded.

    - In server initiated download (Client pull), the client does not send any download request but simply polls to check if the server has any file for this user to be downloaded.
  - Provides means to attach metadata on a transfer operation. This metadata is delivered to the adapter during the session setup time. This metadata could contain information about the transfer. For example, in thin client FT solution, the metadata contains information about the workspace, folder and document.
  - Provides a Java API to write FT Gateway services. For each Gateway service, there exists an Adapter on the FT Server side to service the transfer. Multiple services can be plugged into a single instance of the FT Gateway.
  - Can be ***deployed in either enterprise mode or thin client mode***. In enterprise mode, multiple users can use a single instance of the Gateway to transfer files to/from E2open. An example of this mode is PDM Gateway (PDM service + FT Gateway deployed on the Edgebox). In thin client mode, the session always belongs to a single E2open user. In this mode, it can be deployed on customers' desktop.
  - ***Stickiness to a particular FT Server is only at the transfer session level*** and not at the FT Gateway level. This enables multiple FT Servers to be deployed in a clustered environment.
  - Provides an audit log for all transfer through the FT Gateway.

{ EMBED Visio.Drawing.6 }

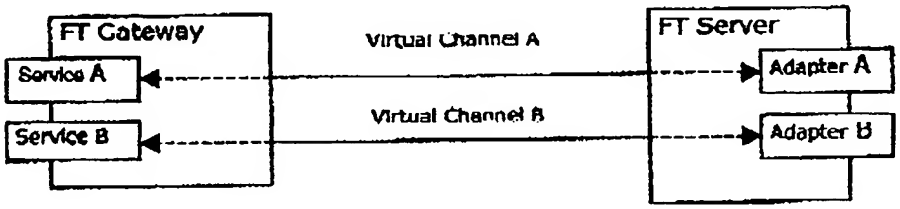
- **File Transfer (FT) Server**:It is installed at E2open hub.
  - Deployed behind E2open's Webseal.
  - A single instance of the FT Server can handle both thin client FT gateway and enterprise FT gateway.
  - Responsible for creating transfer session. Although FT Gateway initiates the transfer, the sessions are created only at the server. All transfers need to have a valid server side transfer session.
  - ***All client side transfer related parameters (QOS parameters like file size, maximum chunk size, time to live etc) are validated at the server side***. This enables the FT server to dictate the maximum size of the file that can be uploaded, maximum size of a file chunk, maximum expiry time, etc.
  - Validate every http transfer request for security.
  - ***Allocates the disk space for the file to be uploaded at the time of creating the transfer session***. If the server runs out of space, it sends a retry time to the client. Enhancement: Since the file server is a shared resource, it would be nice if the FT Server can manage and be configured with a disk quota.
  - ***Re-assembles the chunk dynamically into the file without creating extra physical chunks***. Since the client can send multiple chunks of the same file concurrently, the server is capable of handling chunks that come out of sequence.
  - Handle multiple uploads and downloads concurrently for the same user or for different users.

- Provides a Java API to write FT Server Adapters. Multiple adapters can be plugged into a single instance of the server. An adapter exists for each FT Gateway service.
- For an upload operation, the file transfer is done when the file has been successfully assembled at the server end. But the transfer is completed only when the transfer has been serviced by the adapter. If the server restarts after the transfer has been done but before it could be successfully serviced or the adapter could not successfully service the transfer operation, the server will retry to service the transfer operation after a configurable interval. This will continue till the service operation succeeds or the transfer operation expires. This provides reliability and guarantee of transfer.

4 Design

The File Transfer Service is designed to reliably transfer small and large files over the internet to and from E2Open. The service is based on a service oriented model to support both thin clients (that live on user's desktop) and enterprise clients (that live on backend servers).

The FT Service is designed in a service-adapter model. This model allows the FT service to be used simultaneously for various purposes. Services are deployed on the FT Gateway and the adapters are deployed on the FT Server. For every service, there is a corresponding adapter, thus creating a virtual channel. ***Transfer operations take place within this virtual channel.*** A customized service and adapter can be developed to use the FT service for a specific purpose like PDM Integration. Multiple services/adapters can be deployed on the single instance of the gateway/server



4.1 FT Protocol Messages

- ***EstablishLoginSession***: This message takes the users login Id and login password and attempts to login into E2open via WebSeal. On a successful login, a user session is created and cached in-memory. Depending upon the Gateway configuration, this message may use either the regular E2Open login or CDSSO. Also, as a part of response to this message, ***the server informs the client about the number of concurrent transfer sessions it is allowed***. Currently, this returns a constant number from the server configuration.

- ***TransferContextMessag*** : This message is used by the Gateway to establish a client transfer session with the FT Server. ***QOS management is also achieved with this message***. Important values negotiated at this time are:
    - Maximum File Size.
    - Maximum Chunk Size
    - Maximum Time to live i.e. the duration for which this transfer is valid.
    - Maximum number of threads a client can use for concurrent transfer of chunks.

  The client only recommends the above values. The FT server dictates the final values of these attributes. This requests also contains other transfer information such as user-Id, client task Id, filename, any metadata specified at the time of submission of transfer request by the Gateway service etc. This message uses the SOAP on HTTP transport mechanism. The session attributes are sent in the XML part and the metadata as attachment. On receiving this request, the server
    - Validates the transfer request from the gateway. The server could deny the request on grounds, such as file size is more than the maximum permitted file size, server out of disk space, server is busy etc.
    - The server then tries to validate the transfer request with the corresponding adapter. It is up to the adapter implementation to validate the transfer request. For example, the PDM adapter validates the Seagate Global ID with LDAP and then checks if the user has a valid CM account. Another e.g. – In case of a download operation from CM, the adapter could try to checkout the file for download.
    - On a successful validation, the server creates a transfer session Id for this transfer operation. Also, if the transfer is an upload operation, then the server pre-allocates the space by creating a transfer file for this session. As a part of the response attribute, the server sends the transfer session Id. The client needs to use this session Id in all subsequent requests for this transfer operation.
    - If the validation failed, the server returns an appropriate error. Depending upon error, it may return a retry-Interval after which the gateway can try to resubmit the transfer request.
- ***ChunkFileMessage***: This message does the actual transfer of payload. A ChunkFileMessage object is created for every file chunk to be transferred. This message uses basic HTTP transport mechanism. In an upload transfer, the payload is sent as the request body and in download transfer, the payload is in the response body. The message attributes define the transfer session Id, chunk number, total bytes in the chunk etc. The server on receiving and validating this message
    - For an upload operation, takes the payload and streams it to the transfer file at the right offset. There is no temporary file created.
    - For a download operation, the server takes the chunk bytes from the file and streams them as a part of the response body.
- ***EndTransferMessage***: The gateway sends this message when it determines the transfer has been completed. On receiving this message, the server cleans up the resources associated with the transfer session.

- ***GetResponseMessage***: This message is used by the gateway to poll for a response from the server after an upload operation has been completed. This message is only used only if the gateway service requested a response from the adapter on the completion of the transfer. The gateway polls for the response from the server at a configurable interval (and overridden by the retry-Interval from server). Once a response is received and processed by the gateway service, the gateway sends EndTransferMessage.
- ***GetDownloadMessage***: Only in the enterprise mode, on the request of the service, the gateway polls the server to check if the user has any message that needs to be downloaded. For this, the service has to register the userId with the gateway, for which it has to poll. The polling rate is configurable on the gateway and the server can override it with retry-Interval.
- ***AbortTransferMessage:*** The FT Gateway sends this message, if the transfer needs to be aborted before it has been completed.

4.1.1 FT Transport Mechanism

The FT Service is designed to be secure and uses ***HTTPS as the basic transport mechanism***. The FT Gateway uses a combination of basic HTTP and SOAP over HTTP to communicate with FT Server via WebSeal. The transport protocol is decided by the protocol message. At the protocol layer, the transport mechanism is transparent. Only at the creation of the message object, the object specifies which transport protocol to use. With this model, any of the protocol messages can be switched from basic HTTP to SOAP over HTTP.

4.1.1.1 Basic HTTP Transport

This mechanism makes use of the HTTP protocol to communicate with the server. The attributes for a context are passed as the name-value pairs in the value of the HTTP context header. The values of the attribute names are encoded.

X-E2open-<Context>: <attribute name value pairs>

With this mechanism, multiple context headers can be sent to the server. The services can also set context properties to be sent through to the corresponding adapter.

The message content is sent as the payload of the HTTP request. The payload length is defined in the HTTP Content-Length header. The message definition is defined in the HTTP header Content-Type.

4.1.1.2 SOAP over HTTP Transport

This makes use of SOAP over HTTP to communicate with the server. Each Context is sent as a SOAPBodyElement and the attributes are the XML elements in the SOAPBodyElement.

The message content is sent as the attachment part (SOAP with Attachment).

Currently only the TransferContextMessage uses SOAP over HTTP and all other messages use basic HTTP. The message definition is defined in the SOAPAction header.

Disadvantages of using SOAP over HTTP for all messages

Performance was a key thought in designing FT service and it is for this reason that SOAP was not chosen for all messages. In particular, the performance of the ChunkFileMessage will be greatly reduced in a SOAP over HTTP implementation. This message carries the file chunk as the payload. The chunk size could be 1MB or more. In SOAP context, this would be sent as a SOAP attachment. When SOAP is used with attachments, MIME multipart/related format is used to send this message. To retrieve the attachment, the MIME parser would read the complete chunk byte by byte to reach up to the multipart delimiter. This would degrade the performance significantly. In basic HTTP transport, the chunk length is defined in the Content-Length header and there is no MIME parsing involved in parsing the large chunk data.

In order to use SOAP for this message, one would need to compensate for this shortcoming in MIME parsing. The fix will involve specifying the length of the chunk in the Content-Length header of the MIME body part containing the attachment. The MIME parser will also need to be changed to lookup the length specified in the header and bulk read the bytes to get to the multipart delimiter. In light of this discussion, it should be obvious that using SOAP over HTTP for all messages will involve additional time, effort and changes to the basic MIME parser. The basic HTTP transport mechanism has therefore been chosen for all messages but one (for the exception, see below).

Advantages of using SOAP over HTTP for certain messages

SOAP over HTTP involves some additional overhead and parsing. However, basic HTTP has a 4K limit on the size of the header. The TransferContextMessage passes a significant amount of information including session cookies and context information. In order to accommodate the large amount of information and provide for easy extensibility in the future, SOAP over HTTP has been chosen as the transport mechanism for this message. The message attributes are sent in the XML part and the transfer metadata as attachment. Alternatively, one could have used multipart/mixed, one part defining the session attributes and other the metadata.

4.2 FileTaskInfo persistent cache

The FT Service needs to transfer files in a guaranteed and reliable manner. For this, it needs to manage and maintain the state of transfer and other transfer information across the restarts of the service. FileTaskInfo class is used both by the FT Gateway and FT Server to achieve this.

A FileTaskInfo object is created for each transfer operation. Every instance has a unique taskId. All information by this object is managed in a directory created by the class upon its creation.

- For an upload operation the directory is
  ${WORK_DIR}/<client / server>/<userId>.lft/outbox/<taskId>
- For an download operation the directory is
  ${WORK_DIR}/<client / server>/<userId>.lft/inbox/<taskId>

4.3 Transfer Operations

4.3.1 Upload file into E2Open

The following sequence of steps occur when a file is uploaded into E2open

1. Client initiates an upload file operation. This happens through one of the gateway services, which submit the upload transfer command to the Gateway. At this time the service may also pass some service specific data to gateway to be sent to the corresponding adapter.
2. Gateway establishes a session by sending a *TransferContextMessage* to the server.
3. Once the session is setup, Gateway starts sending *ChunkFileMessages* to the server.
4. After all the chunks are received at the server, the server calls into the adapter with uploadFile in a separate adapter thread.
5. Once the gateway has sent all the chunks and received chunksCommited for all chunks from the server, it does one of the following
   a. If a response is needed from the adapter for the uploaded file, the gateway starts polling the server for the response with *GetMessageResponse* message. Once the response is available to the server, it sends the same to the gateway. Once the gateway has the response it calls into the gateway service with the response and then proceeds to the next step.
   b. If no response is required from the server, the gateway sends EndTransferMessage to the server, cleans up the resources associated with the transfer. The server also cleans up the resources associated with the transfer.

4.3.2 Client initiated download file from E2Open

The following sequence of steps happens to download a file from E2Open.

1. Client initiates a download file operation. This happens through one of the gateway services, which submit the download transfer command to the Gateway. At this time the service may also pass some service specific data to the gateway to be sent on to the corresponding adapter.
2. FT Gateway establishes a session by sending a TransferContextMessage to the server. The adapter prepares the file for being downloaded in a separate adapter thread.
3. The FT Gateway then periodically checks with TransferContextMessage, if the file has been made available at the server to be downloaded. Once the file is available, transfer session setup is complete.
4. If the file for some reason cannot be downloaded, the server denies the download transfer operation to the gateway.
5. Once the session is setup, the gateway starts downloading the file using ChunkFileMessage.
6. After all the chunks are downloaded, the gateway notifies the appropriate service with the downloaded file.

7. After the service has processed the file and IClientService.downloadCompleted returns true, the gateway sends EndTransferMessage to the server. It then cleans up the resources associated with the transfer.

4.3.3 Download file from E2open (push by E2Open)

This feature is supported by Gateway only in the enterprise mode. This feature can be used to push files from E2Open to the clients. For this purpose, the gateway (which resides on the client side) polls the server to check for files to be downloaded. In the current implementation, the files to be pushed to the gateway are put in the directory

${WORK_DIR}/pending/<userId.lft>

1. The gateway service registers with the gateway to poll the server if there are any files to be downloaded for a given user.
2. The Gateway polls the server to check for files to be downloaded with GetDownloadMessage. When the server receives this message, it checks the pending download directory to check if there are any files pending to be downloaded. If any such files exist, then the files are moved to the inbox directory and a transfer session is created for the downloaded operation. The session ids are sent back to the gateway in the response for GetDownloadMessage.
3. On receiving a session Id, the gateway sends a TransferContextMessage for each of the session Ids.
4. The steps after this point are the same as in client-initiated download (steps 3, 4, and 5).

4.4 FT Gateway

The FT Gateway is the client component of FT Service. It is implemented as a standalone multithreaded java application. The gateway can transfer multiple files concurrently (configurable) with multiple threads per transfer (configurable). A FileTaskInfo (persistent cache object) object is created for each operation submitted to the gateway. This object also manages the state of transfer.

A Client task is a task work that is executed by a worker thread. These worker threads are managed by gateway thread pool.

A FileTransferTask (extends ClientTask) is created for each transfer operation. This task basically manages the transfer session (login message, transfer context message, end transfer session). After the session is setup, it creates 'n' ChunkFileTask's, where 'n' is the maximum number of threads allowed for the transfer operation. Each ChunkFileTask creates a ChunkFileMessage and proceeds with the transfer of the file. On completion of the ChunkFileTask, if more chunks are to be transmitted, another ChunkFileTask is created to handle the chunk. Once all the chunks are transmitted, the FileTransferTask sends the EndChunkMessage and cleans up the resources associated with the transfer.

The Gateway is designed to make sure that th file has been transferred reliably and every end point updated with the transfer status. Therefore a retry mechanism has been built into several places.

- If the server is not available, the server will keep retrying, until the transfer's time-to-live expires. The retry interval is configurable.
- If the gateway is stopped or killed while a transfer operation is in progress, on restart the gateway resumes the transfer operation from where it left off.
- If the gateway is stopped or killed after an upload transfer is complete but before it could fetch the response, on restarting it fetches the response and notifies the gateway service with the response.
- If the gateway service returns *false* on IClientService.*uploadCompleted* callback, the gateway will keep retrying till it return true. (This may be of use if the service needs to contact some other process or server to complete this call, and the process/server may not be available).

4.4.1 FT Gateway Services

A FT Gateway service is used to customize FT Service for transfer of files for a specific need. Services are loaded up and initialized when the FT Gateway starts. The services are specified in the services.properties configuration file. The service uploads and downloads files from E2open using the FT Gateway instance.

A FT Gateway Service implements the **com.e2open.ift.client.service.IclientService** interface (defined in Appendix).

4.5 FT Server

The FT Server is implemented as a servlet and can be deployed in any servlet container. Each request from the gateway is validated and processed. It maintains and manages a transfer session for each transfer operation in progress. All sessions make use of FileTaskInfo to manage transformation across restarts. It also manages a thread pool for adapter tasks. All callbacks into adapter are made by in these adapter threads.

4.5.1 FT Server Adapter

A FT Server Adapter is used to customize the FT Service for transfer of files for a specific need. There is an adapter for every gateway service (both the adapter and the service have same name). All the adapters are loaded and initialized when the server starts up. The adapters are defined in the configuration file adapter.properties.

The FT Server adapter implements the interface **com.e2open.ift.server.adaper.IserverAdapter.**

5 Use Case

5.1 Thin FT Client

Thin Client File transfer solution uses thin client service with FT Gateway. This solution is shrink wrapped as an application that gets installed on a users desktop or laptop (currently windows only). The sequence of steps is as follows.

- User logs into E2Open Collaboration Manager. Navigates to a folder and clicks on the file upload/checkin or download/checkout link.
- User starts the Thin FT application. The application can be configured to automatically start when the user logs into Windows.
- User chooses to transfer the file using FT application.
- A hidden signed applet in the browser page submits the transfer request along with metadata about the folder, document etc to the FT application.
- The thin FT service processes this request and submits the transfer task to the FT Gateway. The transfer operation now proceeds in the background.
- If the user chooses to encrypt the document while uploading, the thinFTService encrypts the document with a symmetric key for the CM Workspace from user's local key store. If the user is downloading a encrypted document, the thinFTService will decrypt the document with the appropriate key from the user's local key store. Additional information on this feature can be found in *CM Secured Document Solution*.
- For an upload operation, the file is checked into CM by thinFTAdapter.
- For a download operation, the file is checked out from CM by thinFTAdapter. Once the file is on the client side, the thinFTService moves the file to the destination directory chosen by the user.

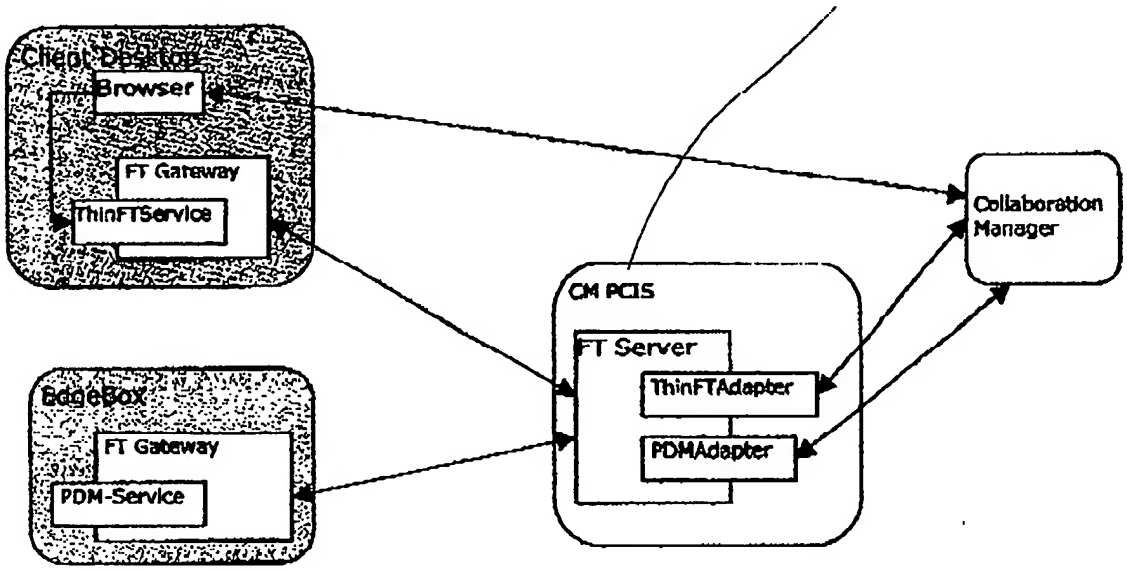
5.2 PDM Integration

PDM Service along with the FT Gateway is deployed on the edge box. The sequence of steps is as follows.

- The PDM service polls a spool directory for files to be transferred into CM.
- The file is packaged as a PCDX document (XML format defined by E2Open).
- The PDM service submits the document to be transferred to the FT Gateway.
- At the transfer session setup time, the PDM adapter verifies the user Id from the document name.
- Once the session is setup, FT Gateway transfers the document to the server.
- FT adapter parses the PCDX document, extracts the file to be checked in and completes the upload operation into CM.
- FT adapter generates a response XML document.
- FT Gateway polls for the response document, retrieves it and delivers it to the PDM service.
- PDM service moves the response document to the response directory to be consumed by customer's application.

For additional information, refer to *Seascape PC-Integration Design* document.

5.3 Deployment Diagram



6 Appendix

6.1 FT Gateway Service Interface

```
/**
 * Interface for the FT Gateway service.
 */
public interface IClientService
{
   /**
```

```
     * Initialize the client service. This method is called into the service when
     * ServiceManager is initalizing this service. At this time the Gateway has not
     * yet stated, and so no transfer related activity happens.
     * @param   pManager        a refernce to the ServiceManager instance.
     * @param   pServiceName    name of this service.
     * @param   pConfigLoc      the config file for this service.
     * @throws  ServiceException the service should throw this exception if it
     *                  could not initialize.
     */
    public void initialize(IServiceManager     pManager,
                           String              pServiceName,
                           String              pConfigLoc) throws ServiceException;

    /**
     * Returns the name of this service
     * @returns name of this service.
     */
    public String getServiceName();


    /**
     * Start the service. This is a callback from the ServiceManager into this
     * service when this service is being started. The service implements this
     * method the procedure to start this service.
     * @throws ServiceException the service throws this exception if it failed
     *                  to start
     */
    public void start() throws ServiceException;

    /**
     * Callback from FT Gateway into the service with the tasklist at initialize
     * time.
     * @param pTaskList list of task pending at the initialization time. This is
     *              task list which the FT Gateway is going to process.The service
     *              does not need to resubmit the task.
     */
    public void initialTaskList(ArrayList pTaskList);

    /**
     * This callback is received by this service when the FT Gateway is
     * shuting down. Here the service needs to implement procedure for shutting
     * down the service, cleanup etc.
     */
    public void shutdown();
```

```
/**
 * This callback is received by a service if the transfer
 * operation was denied by the FT Server (E2open) for some reason.
 * @param pTaskId          the taskId of the transfer operation.
 * @param pFilename        the name of the file to be transfered.
 * @param pReasonStr       description of the failure.
 * @param pReason          error code for transfer denial.
 **/
public void transferDenied(String  pTaskId,
                           String  pFilename,
                           String  pReasonStr,
                           int     pReson);


/**
 * This callback invoked on completion of upload task.
 * @param pTaskId          the taskId of the transfer operation.
 * @param pTransferFile    complete name of the file in transfer.
 * @param pResponseFile    the response file
 * @param pResponseStatus  status.
 *
 * @returns the service should return true if the service was able to
 *          handle the callback. On completion of this callback,
 *          the FT Gateway will mark this task serviced and will clean
 *          up the task. If the service returned false, then the Gateway
 *          will try to service the call again at a later time.
 */

public boolean uploadCompleted(String pTaskId,
                               String pTransferFile,
                               String pServiceData,
                               String pResponseFile,
                               String pResponseStatus);


/**
 * Callback received on the completion of a download.
 * @param pTaskId          the taskId of the transfer operation.
 * @param pFilename        the final complete name of the file downloaded.
 * @param pTransferFilename the physical file that represents the downloaded file.
 * @param pServiceData     service data associated with this taskId, if any.
 *                         This is the cache data set by the service
 *                         for this task.
 *
```

```
     * @returns the service should return true if the service was able to
     *          handle the callback. On completion of this callback,
     *          the FT Gateway will mark this task serviced and will clean
     *          up the task. If the service returned false, then the Gateway
     *          will try to service the call again at a later time.
     */
    public boolean downloadCompleted(String pTaskId,
                                     String pFilename,
                                     String pTransferFilename,
                                     String pServiceData);

    /**
     * Get the copyFunctor for this task.
     * For a file being uploaded, the service can specify if the file needs to be
     * copied in FT Gateway cache for transfer. If the file needs to be copied to the
     * cache, then the Gateway calls into the service to get CopyFunctor.
     *
     * @param   the taskId for which this callback is made.
     * @returns copyFuntor used to make a copy of the file. If the service want to
     *                  use the default mode of copy (plain file copy), it should
     *                  return null. The service can implement ICopyFunctor to
     *                  to specialize the the copy function, i.e. compress the file
     *                  while copy, encrypt the file while copy etc.
     */
    public ICopyFunctor getCopyFunctor(String pTaskId);

}
```

6.2 Server Adapter Interface

```
/**
 * Inteface to implement adapters for LFT Server.
 * Each adapter has a name. The name of the adapter is passed as the name
 * in the application context.
 */
public interface IServerAdapter
{
   /**
    *initialize the adapter
    * @param pName name of this adapter. This name is picked up from
```

```
*              the adapter list file adapter.properties
* @param pConfigFile the config file for this adapter.
* @param pAdapterManager Manager interface that the adpater could cache for
*                        for later use.
*
*/
public void initialize(String              pName,
                       String              pConfigFile,
                       IAdapterManager     pAdapterManager)
  throws AdapterException;

/** returns the name of this adapter. */
public String getAdapterName();

/** shutdown the adapter. This method is called before the server is going down */
public void shutdown();


/**
 * The call is made the the server into the adapter to validate the request
 * for a transfer operation. This call is made before the session is setup
 * for the transfer.
 * @param  pSessionId - taskId for the transfer session to be established
 * @param  pFilename - name of the file in transfer
 * @param  pFileSize - size of the file.
 * @returns true if the session can be setup.
 */
public boolean validateUpload(String pSessionId,
                              String pFilename,
                              long   pFileSize)
  throws AdapterException;

public boolean validateDownload(String pSessionId,
                                String pFilename)
  throws AdapterException;


/**
 * This method called in by the server into the adapter after the file has been
 * uploaded on the server. The adapter needs to handle the file in this
 * method.
 * @param pSessionId    the transfer Id associated with this transfer
 * @param pTaskSubmitTime the time at which this task was submitted to the
 *                        gateway
 * @param pTransferFilename the transfe file name of uploaded file.
```

```
     * @param pFilename the uploaded file name
     * @param pServiceData data sent by the lft service (client side)
     * @param pAdapterData  data set by this adapter for this task.
     *                This is the data that the adapter could have set while
     *                validating the transfer request.
     * @returns true if the adapter handled the upload successfully else return false.
     */

    public boolean uploadFile(String        pSessionId,
                              long          pTaskSubmitTime,
                              String        pTransferFilename,
                              String        pFilename,
                              String        pServiceData,
                              String        pAdapterData)
      throws AdapterException;


    /**
     * @param pSessionId       the transfer Id associated with this transfer
     * @param pTransferFilename the transfe file name of uploaded file.
     * @param pFilename the uploaded file name
     * @param pServiceData data sent by the lft service (client side)
     * @param pAdapterData  data set by this adapter for this task.
     *                This is the data that the adapter could have set while
     *                validating the transfer request.
     * @returns true if the adapter handled the upload successfully else
     *                return false.
     */

    public boolean downloadFile(String        pSessionId,
                                long          pTaskSubmitTime,
                                String        pTransferFilename,
                                String        pFilename,
                                String        pServiceData,
                                String        pAdapterData)
      throws AdapterException;


    /**
     */
    public boolean downloadComplete(String      pSessionId,
                                    String      pFilename,
                                    String      pAdapterData);

    /**
```

```
     * callback on transfer being aborted by the client.
     */
    public void transferAborted(String      pSessionId,
                                String      pFilename,
                                String      pAdapterData);

}
```

6.3 ServiceManager Interface

```
/**
 * Interface for ServiceManager.
 */
public interface IServiceManager
{
    /**
     * A service invokes this function to upload a file to E2open. This method
     * only submits a upload operation to the FT Gateway.
     * @pService        the service invoking this function.
     * @pUserId         the userId used to authenticate this transfer
     * @pPasswd         the password used for authenticating pUserId.
     * @pFilename       complete path to the file that is to be uploaded.
     * @pNeedsResponse  does this upload needs a response from the
     *                  corresponding adapter.
     * @pCopyFile       If set true, the the Gateway will make a copy of the
     *                  file for transfer. If false, the the same physical file
     *                  will be used for transfer.
     * @returns         the taskId for the transfer operation.
     *@throws LFTException if the upload transfer could not be submitted.
     */
    public String uploadFile(IClientService     pService,
                             String             pUserId,
                             String             pPasswd,
                             String             pFilename,
                             boolean            pNeedsResponse,
                             boolean            pCopyFile)
      throws LFTException;


    /**
     * A service invokes this function to upload a file to E2open. This method
     * only submits a upload operation to the FT Gateway.
     * @pService        the service invoking this function.
```

```
     * @pUserId        the userId used to authenticate this transfer
     * @pPasswd        the password used for authenticating pUserId.
     * @pFilename      complete path to the file that is to be uploaded.
     * @pAppData       application data that needs to sent to the adapter along
     *                 with this upload operation.
     * @pNeedsResponse  does this upload needs a response from the
     *                 corresponding adapter.
     * @pCopyFile      if set true, the the Gateway will make a copy of the
     *                 file for transfer. If false, the the same physical file
     *                 will be used for transfer.
     * @returns        the taskId for the transfer operation.
     *@throws LFTException if the upload transfer could not be submitted.
     */
    public String uploadFile(IClientService    pService,
                             String            pUserId,
                             String            pPasswd,
                             String            pFilename,
                             String            pAppData,
                             boolean           pNeedsResponse,
                             boolean           pCopyFile)
      throws LFTException;


    /**
     * Invoked by service to download a file from the service.
     */
    public String downloadFile(IClientService    pService,
                               String            pUserId,
                               String            pPasswd,
                               String            pFilename)
      throws LFTException;

    /**
     * Submit a request to download file from the server. A service uses this
     * method to initiate a download request from the server.
     * @param   pService       the service submitting this request.
     * @param   pUserId        the UserId for the download operation.
     * @param   pPasswd        the passwd for this download operation.
     * @param   pFilename      the local file name for the file to be downloaded.
     * @param   pAppData       the application meta data for this download op.
     *                         If set, this meta data is sent to the FT Server.
     *                         FT Server delivers this meta data to the adapter
     *                         associated with this service, at the time when
     *                         the download session is being established.
     *
```

```
     * @throws  LFTException    if the request submission failed for some reason.
     */
    public String downloadFile(IClientService    pService,
                               String            pUserId,
                               String            pPasswd,
                               String            pFilename,
                               String            pAppData)
      throws LFTException;


    /**
     * Invoked by a service to abort a transfer.
     * @param pService the service requesting this operation.
     * @param pTaskId the taskId of the task to be aborted.
     * @returns true if the task was aborted with success.
     * @throws LFTException if an invalid/inactive taskId was provided.
     */
    public boolean abortTransfer(IClientService pService,
                                 String         pTaskId)
      throws LFTException;

    /**
     * Suspend the task with the task Id.
     * @parma pTaskId   taskId of the task to be suspended.
     * @return true if the task was suspended with success.
     * @throws LFTException if an invalid/inactive taskId was provided.
     */
    public boolean suspendTransfer(IClientService pClientService,
                                   String pTaskId)
      throws LFTException;

    /**
     * Resume a suspended file tmasfer operation
     * @parma pTaskId   taskId of the task to be resumed.
     * @return true if the task was resumed with success.
     * @throws LFTException if an invalid/inactive taskId was provided.
     */
    public boolean resumeTransfer(IClientService pClientService,
                                  String pTaskId)
      throws LFTException;

    /**
     * In enterprise mode, a service can register with the FT Gateway to poll
     * for files to be downloaded from the server. This is useful if E2Open
     * wants to push files to the Customers. The FT Gateway will poll at regular
```

```
     * (configurable) interval to check if there are files that needs to be
     * downloaded.
     * @param   pUserId        the userId for whom to poll for download.
     * @param   pPasswd        the password for this userId.
     */
    public void downloadPollForUser(String     pUserId,
                                    String     pPasswd);

    /**
     * Shutdown the complete gateway.
     */
    public void shutdownGateway();


    /**
     * Set service data for a task. A Service can set some service data for a
     * particular task. The data can be retreived with getServiceData method.
     * Once set, this information is available to the service across restarts.
     *
     * @param pService         the service that wants to set the data.
     * @param pUserId          the UserId associated with the task.
     * @param pTaskId          the taskId on which this data is be set.
     * @param pServiceData     the data that needs to be cached for this task
     *
     * @throws LFTException    if the data could not be set.
     *
     */
    public void setServiceData(IClientService   pService,
                               String           pUserId,
                               String           pTaskId,
                               String           pServiceData)
      throws LFTException;

    /**
     * get the service date set by this service for a specific user and
     * specific task
     * @param pService         the service that wants to get the data.
     * @param pUserId          the UserId associated with the task.
     * @param pTaskId          the taskId on which this data was be set.
     *
     * @throws LFTException    if the data could not be retreived.
     */
    public String getServiceData(IClientService pService,
                                 String         pUserId,
                                 String         pTaskId)
```

```
    throws LFTException;

/**
 * Get TaskInforamtion object associated with a particular task.
 *
 * @param pUserId          the UserId associated with the task.
 * @param pTaskId          the taskId whose FileTaskInfo is be retreived.
 *
 */
public IFileTaskInfo       getFileTaskInfo(String pUserId,
                                           String pTaskId)
    throws LFTException;


/**
 * Get list of all submitted task.
 */
public ArrayList getAllSubmittedTasks();


/**
 * set the transfer progress listener for a specific transfer
 * A service can register a Progress Listener to get information on transfer
 * progress.
 *
 * @param pService  the service registering the listener.
 * @param pListener the listener to be registered.
 *
 * @throws LFTException
 */
public void setTransferProgressListener(IClientService      pService,
                                        ITransferProgressListener pListener)
    throws LFTException;

/**
 * Register the exception listener for the service.
 * A service can register a Exception Listener to get exceptions on the task.
 *
 * @param pService  the service registering the listener.
 * @param pListener the listener to be registered.
 *
 * @throws LFTException
 *
 */
```

```
    public void registerExceptionListener(IClientService      pService,
                                          IExceptionListener  pListener)
      throws LFTException;

}
```

6.4 AdapterManager Interface

```
public interface IAdapterManager
{
   public static final int    STATUS_NORESPONSE = 0;
   public static final int    STATUS_SUCCESS = 1;
   public static final int    STATUS_ERROR   = 2;

   /**
    * A adapter can use this method to set some data on the taskId. This data
    * will be passed back to the adapter in the IServerAdapter.uploadFile and
    * IServerAdapter.downloadFile callbacks.
    * @param pAdapter   the adapter intance that is invoking this method.
    * @param pSessionId   the taskId of the transfer for which this method is
    *                called.
    * @param pData     The adapter data that needs to be persisted for this task.
    */

   public void setAdapterTaskData(IServerAdapter pAdapter,
                                  String         pSessionId,
                                  String         pData)
     throws LFTException;


   /**
    * A adapter can use this method to set some data on the taskId. This data
    * will be passed back to the adapter in the IServerAdapter.uploadFile and
    * IServerAdapter.downloadFile callbacks.
    * @param pAdapter   the adapter intance that is invoking this method.
    * @param pSessionId   the taskId of the transfer for which this method is
    *                called.
    * @param pResponseFilename  The response filename for this request, if any.
    */
   public void setTaskResponseFile(IServerAdapter pAdapter,
                                   String         pSessionId,
                                   String         pResponseFilename,
                                   int            pStatus)
```

```
        throws LFTException;

    public String submitFileForDownload(IServerAdapter pAdapter,
                                        String      pToUserId,
                                        String      pFilename)
        throws LFTException;
}
```